



Clase 7. Python

Repaso clase 6

***RECUERDA PONER A GRABAR LA
CLASE***





OBJETIVOS DE LA CLASE

- Utilizar funciones avanzadas de cadenas, listas, conjuntos y diccionarios

CRONOGRAMA DEL CURSO

Clase 6



Conjuntos y Diccionarios



SETS



DICTS

Clase 7



Métodos de colecciones



COLECCIONES 1



COLECCIONES 2

Clase 8



Manejo de archivos y datos



MI MASCOTA



CURIOSOS POR LA
INFORMACIÓN

CADENAS

Upper



Esta función integrada sirve para hacer que se devuelva la misma cadena pero con sus caracteres en **mayúscula**, usando el método **upper()**. Se escribe como: *string.upper()*

```
>>> cadena = "Hola Mundo"
>>> cadena.upper()
"HOLA MUNDO"
>>> "hola amigo!".upper()
"HOLA AMIGO!"
```



Lower

Ya vimos cómo convertir a mayúsculas, pero también es útil **convertir una cadena de caracteres a minúsculas**, usando el método **`lower()`**. Se escribe como: *string.lower()*

```
>>> cadena = "Hola Mundo"
>>> cadena.lower()
'hola mundo'
>>> "HoLa AmIgO!".lower()
'hola amigo!'
```



Capitalize

Esta función integrada sirve para hacer que se devuelva la misma cadena pero con su **primer** carácter en **mayúscula** y **el resto** de caracteres hacerlos **minúscula**, usando el método ***capitalize()***. Se escribe como: *string.capitalize()*

```
>>> cadena = "Hola Mundo"
>>> cadena.capitalize()
"Hola mundo"
>>> "HoLa AmIgO!".capitalize()
"Hola amigo!"
```




Title

Esta función integrada sirve para hacer que se devuelva la misma cadena pero con el **primer** carácter de cada palabra en **mayúscula** y el resto de caracteres hacerlos **minúscula**, usando el método ***title()***. Se escribe como: *string.title()*

```
>>> cadena = "hOLA mUNDO"  
>>> cadena.title()  
"Hola Mundo"  
>>> "HoLa AmIlgO!".title()  
"Hola Amigo!"
```

Count



Si necesitamos saber **cuántas veces aparece una subcadena** dentro de la misma **cadena**, usando el método **count()**. Se escribe como:

string.count()

```
>>> cadena = "hOLa mUNDO esta  
cadena tiene muchas a"  
>>> cadena.count("a")  
6  
>>> "HoLa amigo como estas  
amigo!".count("amigo")  
2
```

Find



Si necesitamos averiguar el índice en el que aparece una **subcadena** dentro de la misma **cadena**, usamos el método **find()**. Se escribe como: *string.find()*. Si no encuentra la cadena devuelve un **-1**.

```
>>> cadena = "hOLa mUNDO  
esta cadena tiene muchas a"  
>>> cadena.find("esta")  
11  
>>> "HoLa amigo como estas  
amigo!".find("chau")  
-1
```



Rfind

Es exactamente igual al método **find()** lo diferencia en que **rfind()** devuelve el índice pero de la última ocurrencia de la subcadena, es decir, la última vez que aparece en la cadena. Se escribe como: `string.rfind()`. Si no encuentra la cadena devuelve un **-1**.

```
>>> "HoLa amigo como estas  
amigo!".find("amigo")  
5  
>>> "HoLa amigo como estas  
amigo!".rfind("amigo")  
22
```



Split

Esta función integrada sirve para devolver una lista con la cadena de caracteres separada por cada índice de la lista.

```
>>> cadena = "hOLA mUNDO"  
>>> cadena.split()  
["hOLA", "mUNDO"]  
>>> "HoLa amigo como estas  
amigo!".split("amigo")  
["HoLa", "como ", "estas ", "!" ]
```

Se escribe como:

`string.split("cadena_a_separar")`. Si no se indica alguna cadena para separar se separa por **"espacios"**.



Join

Esta función integrada sirve para devolver una cadena separada a partir de una especie de separador. Se escribe como: “**separador**”.**join**(“cadena”).

```
>>> cadena = "Hola mundo"
>>> ",".join(cadena)
"H,o,l,a, ,m,u,n,d,o"
>>> " ".join(cadena)
"H o l a  m u n d o"
```

Nota: Si no se especifica el separador nos devuelve un error



Strip

Esta función integrada sirve para devolver una cadena borrando todos los caracteres delante y detrás de la cadena. Se escribe como:

cadena.**strip**("caracter_a_borrar").

```
>>> cadena = "-----Hola  
mundo-----"  
>>> cadena.strip("-")  
"H,o,l,a, ,m,u,n,d,o"  
>>> "          Hola mundo  
.strip()  
"Hola mundo"
```

Nota: Si no se especifica el carácter elimina los espacios

CODER HOUSE



Replace

Esta función integrada sirve para devolver una cadena reemplazando los subcaracteres indicados.

Se escribe como: `cadena.replace("caracter_a_reemplazar", "caracter_que_reemplaza")`. También podemos indicar cuantas veces lo reemplazaremos utilizando un índice.

Replace



```
>>> cadena = "Hola mundo"
>>> cadena.replace("o", "0")
"H0la mund0"
>>> "Hola mundo mundo mundo mundo mundo".replace(' mundo', '', 4)
"Hola mundo"
```

Nota: En el último reemplazamos mundo 4 veces por un sólo carácter vacío

LISTAS



Clear

Como vimos en listas, para “eliminar” todos los elementos de una lista podíamos hacer `lista = []`, sin embargo, también podemos usar **`clear()`** para vaciar todos los ítems de la lista. Se escribe como: `lista.clear()`

```
>>> letras = ['a', 'b', 'c', 'd', 'e', 'f']
>>> letras.clear()
[]
```



Extend

Como vimos en las listas, podemos sumar una lista con otra lista de la siguiente forma:

```
>>> numeros = [1,2,3,4]
>>> numeros + [5,6,7,8]
```

Pero también podemos hacer uso de `extend` ya que une una lista con otra. Se usa como `lista.extend(otra_lista)`

```
>>> lista1 = [1,2,3,4]
>>> lista2 = [5,6,7,8]
>>> lista1.extend(lista2)
```



Insert

```
>>> lista = [1,2,3,4,5]
>>> lista.insert(0, 0)
[0,1,2,3,4,5]
>>> lista2 = [5,10,15,25]
>>> lista2.insert(-1, 20) # Anteúltima posición
[5,10,15,20,25]
>>> lista3 = [5,10,15,25]
>>> n = len(lista3)
>>> lista3.insert(n, 30) # Última posición
[5,10,15,25,30]
```

Esta función integrada se usa para **agregar un ítem a una lista, pero en un índice específico**. Se escribe como:

lista.insert(posición, ítem).



Reverse

Esta función integrada sirve para **dar vuelta una lista**. Se escribe como:
`lista.reverse()`

```
>>> lista = [1,2,3,4]
>>> lista.reverse()
[4,3,2,1]
```

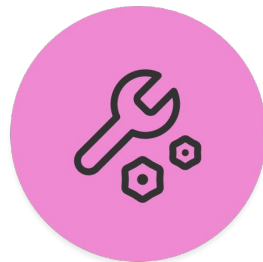
Nota: Las cadenas no tienen la función `reverse`, pero se puede simular haciendo una conversión a lista y después usando el `join`



Sort

Esta función integrada sirve para **ordenar una lista** automáticamente por valor, **de menor a mayor**. Se escribe como: **lista.sort()**. Si ponemos el argumento **reverse=True** la lista se ordenará de **mayor a menor**.

```
>>> lista = [5,-10,35,0,-65,100]
>>> lista.sort()
[-65, -10, 0, 5, 35, 100]
>>> lista.sort(reverse=True)
[100, 35, 5, 0, -10, -65]
```



COLECCIONES 1

Transforma el texto

Tiempo estimado: 20 minutos en caso de no terminar continuarlo

en casa

CODER HOUSE



COLECCIONES 1

Tiempo estimado: 20 minutos

Utilizando todo lo que sabes sobre cadenas, listas y sus métodos internos, transforma este texto:

gordon lanzó su curva&strawberry ha fallado por un pie! -gritó Joe Castiglione&dos
pies -le corrigió Troop&strawberry menea la cabeza como disgustado... -agrega el
comentarista



COLECCIONES 1

Transforma el texto en:

Gordon lanzó su curva...

- Strawberry ha fallado por un pie! -gritó Joe Castiglione.
- Dos pies le corrigió Troop.
- Strawberry menea la cabeza como disgustado... -agrega el comentarista.

Lo único prohibido es modificar directamente el texto



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

CONJUNTOS



Copy

Esta función integrada sirve para hacer que se devuelva una **copia** de un set.

Se escribe como: **`set.copy()`**

```
>>> set1 = {1,2,3,4}
>>> set2 = set1.copy()
>>> print(set2)
{1,2,3,4}
```



isdisjoint

Esta función comprueba si el **set es distinto** a otro set, es decir, si no hay ningún ítem en común entre ellos. Se escribe como: **set1.isdisjoint(set2)**

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.isdisjoint(set2)
False
```

Nota: Devuelve False por que set1 y set2 comparten el 3



issubset

Esta función comprueba si el **set es subset de otro set**, es decir, si todos sus ítems están en el otro conjunto. Se escribe como: ***set1.issubset(set2)***

```
>>> set3 = {-1,99}
>>> set4 = {1,2,3,4,5}
>>> set3.issubset(set4)
False
```

Nota: Devuelve False por que set3 no está todo dentro de set4



issuperset

Esta función es muy similar al `issubset`, la diferencia es que esta comprueba si el set es contenedor de otro set, es decir, si contiene todos los ítems de otro set.

Se escribe como: **`set1.issuperset(set2)`**

```
>>> set5 = {1,2,3}
>>> set6 = {1,2}
>>> set5.issuperset(set6)
True
```


Unión



Esta función une un set con otro, y devuelve el resultado en un nuevo set. Se escribe como: `set1.union(set2)`

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.union(set2)
{1,2,3,4,5}
```

Difference



Esta función encuentra todos los elementos no comunes entre dos set, es decir, nos devuelve un set de ítems diferentes entre cada set.

Se escribe como: `set1.difference(set2)`

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.difference(set2)
{1,2}
```

Nota: Acá devuelve 1 y 2 por qué le pregunta básicamente “que tengo de diferente al set2?”



difference_update

Similar al **difference**, pero esta función nos guarda los ítems distintos en el set originales, es decir, le **asigna como nuevo valor los ítems diferentes.**

Se escribe como: **set1.difference_update(set2)**

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.difference_update(set2)
>>> print(set1)
{1,2}
```

Nota: Ahora set1 vale {1,2} ya que es la diferencia que tenía con set2



intersection

Esta función devuelve un set con todos los elementos **comunes** entre dos set, es decir, nos devuelve un set de ítems **iguales** entre cada set.

Se escribe como: `set1.intersection(set2)`

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.intersection(set2)
{3}
```



intersection_update

Es exactamente igual al intersection, pero esta función actualiza el set original, es decir, le **asigna como nuevo valor los ítems en común**. Se escribe como:

set1.intersection_update(set2)

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.intersection_update(set2)
>>> print(set1)
{3}
```

Nota: Ahora set1 vale los ítems en común
con set2

DICCIONARIOS

get



La función `get` sirve para poder buscar un elemento a partir de su **key**, en el caso de no encontrar devuelve un valor por defecto que le indicamos nosotros.

Se escribe como: `dict.get(key, "valor por defecto")`

```
>>> colores = { "amarillo":"yellow", "azul":"blue", "verde":"green" }
>>> colores.get("rojo", "no hay clave rojo")
"no hay clave rojo"
>>> colores.get("amarillo", "no hay clave amarillo")
"yellow"
```

keys



La función **key** sirve para poder traer todas las claves de un diccionario en el caso de desconocerlas. Se escribe como: **dict.keys()**

```
>>> colores = { "amarillo":"yellow",  
                "azul":"blue", "verde":"green" }  
>>> colores.keys()  
dict_keys(['amarillo', 'azul', 'verde'])
```




values

La función **values** es similar a keys, pero esta sirve para poder traer todos los valores de un diccionario. Se escribe como: **dict.values()**

```
>>> colores = { "amarillo":"yellow",  
                "azul":"blue", "verde":"green" }  
>>> colores.values()  
dict_values(['yellow', 'blue', 'green'])
```

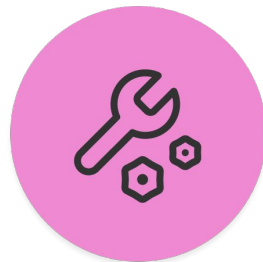
items



```
>>> colores = { "amarillo":"yellow", "azul":"blue", "verde":"green"
}
>>> colores.items()
dict_items([('amarillo', 'yellow'), ('azul', 'blue'), ('verde', 'green')])

>>> for clave, valor in colores.items():
    print(clave, valor)
amarillo yellow
azul blue
verde green
```

La función **items** es similar a keys y values, pero esta crea una lista con clave y valor de los ítems de un diccionario. Se escribe como: **dict.items()**



COLECCIONES 2

Realizar las instrucciones sin modificar la lista original

Tiempo estimado: 30 minutos en caso de no terminar continuar en
casa

CODER HOUSE



COLECCIONES 2

Tiempo estimado: 30 minutos

A partir de una lista realizar las siguientes tareas sin modificar la lista original:

1. Borrar los elementos duplicados
2. Ordenar la lista de mayor a menor
3. Eliminar todos los números impares
(for ---- if (%2==1) ---- pop, remove)
4. Realizar una suma de todos los números que quedan
(sum(lista))



COLECCIONES 2

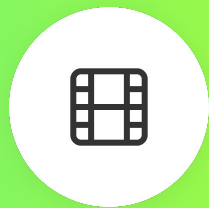
5. Añadir como primer elemento de la lista la suma realizada `insert(0, suma)`
6. Devolver la lista modificada
7. Finalmente, después de ejecutar la función, comprueba que la suma de todos los números a partir del segundo, concuerda con el primer número de la lista

```
lista = [29, -5, -12, 17, 5, 24, 5, 12, 23, 16, 12, 5, -12, 17]
```

Nota: Recuerda que para sumar todos los números de una lista puedes usar **sum**

¿PREGUNTAS?





***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***



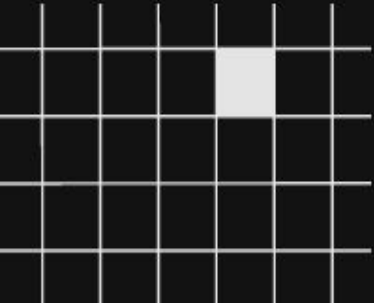
- Artículo: [Cadenas](#)
- Artículo: [Listas](#)
- Artículo: [Tuplas](#)
- Artículo: [Diccionarios](#)
- Artículo: [Conjuntos](#)
- [EjemploClase](#)





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Listas
 - Tuplas
 - Anidación
 - Transformación de colecciones
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE